****

软 件 工 程 实 验

软件产品设计与实现

基于Scapy的分析与扩展

Version 1.4

实 验 小 组 B组

小 组 成 员 陈鸿超（SY1806214）

李铎坤（SY1806219）

刘 颖（SY1806418）

袁梦阳（BY1806157）

编制时间： 2019年4月

版本变更历史

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 版本 | 提交日期 | 编制人 | 修改说明 | 审核人 | 版本说明 |
| v1.4 | 2019/04/24 | 袁梦阳 | 补充第四章图片 |  | 增加时序图 |
| v1.3 | 2019/04/24 | 刘颖 | 修改图4.1，增加图4.2 | 袁梦阳 | 时序图修改 |
| v1.2 | 2019/04/18 | 陈鸿超 | 修改图5.1、5.3 | 刘颖 | 流程图修改 |
| v1.1 | 2019/04/18 | 刘颖 | 修改了4.3、4.4 | 陈鸿超 | 完善扩展功能设计 |
| v1.0 | 2019/04/17 | 李铎坤 | 初版的5.4、6.2  格式调整 | 刘颖 | 初版完成 |
| v0.3 | 2019/04/17 | 陈鸿超 | 初版的5.1、5.2、5.3、6.1 | 李铎坤 | 初版 |
| v0.2 | 2019/04/17 | 刘颖 | 初版的1、2、4.3、4.4 | 袁梦阳  李铎坤 | 初版 |
| v0.1 | 2019/04/17 | 袁梦阳 | 初版的2、4.1、4.2 | 李铎坤 | 初版 |

目录

[版本变更历史 I](#_Toc6511639)

[1 引言 1](#_Toc6511640)

[1.1 编写目的 1](#_Toc6511641)

[1.2 定义 1](#_Toc6511642)

[1.3 参考资料 1](#_Toc6511643)

[2 总体设计 2](#_Toc6511644)

[2.1 需求规定 2](#_Toc6511645)

[2.2 硬件运行环境 2](#_Toc6511646)

[2.3 软件运行环境 2](#_Toc6511647)

[2.4 编程语言与工具 2](#_Toc6511648)

[3 系统架构 3](#_Toc6511649)

[4 扩展功能设计 4](#_Toc6511650)

[4.1 pcap文件切分功能设计 4](#_Toc6511651)

[4.2 获取报文协议类型功能设计 4](#_Toc6511652)

[4.3 监听持久化 5](#_Toc6511653)

[4.4 会话提取功能 6](#_Toc6511654)

[5 实现方案 7](#_Toc6511655)

[5.1 文件切分功能 7](#_Toc6511656)

[5.1.1 兼容模式 7](#_Toc6511657)

[5.1.2 极速模式 9](#_Toc6511658)

[5.2 监听持久化功能 11](#_Toc6511659)

[5.3 协议类型获取功能 12](#_Toc6511660)

[5.4 报文会话提取模块 14](#_Toc6511661)

[5.4.1 基本报文会话提取流程实现 14](#_Toc6511662)

[5.4.2 分批会话提取流程实现 15](#_Toc6511663)

[6 集成方案 17](#_Toc6511664)

[6.1 集成方式 17](#_Toc6511665)

[6.2 验证样例 17](#_Toc6511666)

# 引言

## 编写目的

本软件设计说明书针对需求分析阶段提出的Scapy扩展功能进行描述。

本文档是开发者前期开发的重要依据，是为了明确所开发的扩展模块的具体功能而设计的实现方案。除此之外，本文档也是后期交付时软件是否符合标准的一个重要参考。

## 定义

表 1.1 术语与缩略词表

|  |  |
| --- | --- |
| 术语 | 说明 |
| IP | 互联网协议地址，是分配给用户上网使用的网际协议的设备的数字标签 |
| pcap | pcap 是packet capture 的缩写，pcap文件格式是常用的数据报存储格式 |
| 报文 | 报文(message)是网络中交换与传输的数据单元，即站点一次性要发送的数据块 |
| 端口 | 是一种经由软件创建的服务，在一个计算机操作系统中扮演通信的端点 |
| 端口监听 | 端口监听是指对客户端(个人机器)所操作的一种信息记录，还用于实现对共享目录访问的监测和控制 |
| 接口 | API，是一些预先定义的函数，目的是提供应用程序与开发人员基于某软件或硬件得以访问一组例程的能力，而又无需访问源码，或理解内部工作机制的细节 |
| SMTP | 简单邮件传输协议(Simple Mail Transfer Protocol, SMTP)，是在Internet传输email的事实标准。 |
| HTTP | 超文本传输协议(HTTP，HyperText Transfer Protocol)，是用于从WWW服务器传输超文本到本地浏览器的传输协议。 |
| Tcpdump | tcpdump是Linux系统中功能强大的网络数据采集分析工具。 |
| BPF | 柏克莱封包过滤器(Berkeley Packet Filter，缩写 BPF)，是类Unix系统上数据链路层的一种原始接口，提供原始链路层封包的收发。 |
| WinPcap | WinPcap是Windows系统中提供底层网络接口访问的工具。 |

## 参考资料

1. 吕云翔，王昕鹏，邱玉龙.软件工程——理论与实践.北京：人民邮电出版社，2012.
2. 需求规格说明书2.4.docx
3. 虞平. 一种面向会话的入侵检测模型[D].东南大学,2004.

# 总体设计

## 需求规定

本项目基于Scapy开源项目进行扩展，面向人群为计算机网络方面的开发与测试人员，主要目的是便于用户快捷地进行报文的获取、解析和发送等操作。为了满足此需要，本项目考虑实现一个更全面、快速的报文操作工具。通过对Scapy工具的扩展，在Scapy开源项目中扩展出开发人员需求但却缺少的功能，使得Scapy更加易用与全面。

## 硬件运行环境

1. CPU：Intel i5-4590及以上，AMD Ryzen 3 1200及以上
2. 内存：4G内存及以上（系统最低2GB，Scapy报文解析较占用内存空间）
3. 硬盘：20G硬盘及以上（Windows 7 64位安装最低配置）

## 软件运行环境

1. 操作系统：
   * 1. Windows 7及以上： 已安装Npcap/WinPcap，
     2. Ubuntu 16.04及以上： 已安装tcpdump
2. Anaconda（可创建多版本Python虚拟环境），Python 3.4+
3. 第三方Python库：matplotlib，pyx，vpython，cryptography等

## 编程语言与工具

1. 编程语言：Python
2. 编程工具：pycharm

# 系统架构

作为一个数据包处理工具，Scapy的系统架构比较简单，主要由底层系统、基础模块、应用模块三部分组成。其中，底层系统是操作系统本身自带的功能，基础模块和应用模块则是由Scapy所实现。Scapy的系统架构如图 3.1所示。



图 3.1 系统架构图

基础模块是应用模块的实现基础，对于用户而言是不可见的，主要包括环境配置模块、插件调用模块、网络标准模块、协议格式模块。其中环境配置模块主要负责获取运行环境信息，初始化其他模块所需的配置参数；插件调用模块主要负责调用各种插件，比如tcpdump、WinPcap等；网络标准模块主要负责封装计算机网络领域所使用的编码、传输、解码标准；协议格式模块主要封装了Scapy所支持的所有协议的具体格式，包括所有协议字段名称、数据类型、大小、位置等信息。

应用模块是用户可以直接使用的模块，主要包括数据导入模块、数据导出模块、数据展示模块、数据解析模块、数据构造模块、端口监听模块、数据传输模块。这些模块实现了用户进行数据包处理所需的各种功能，包括数据获取、数据构造、数据存储、数据发送和数据展示等。

我们在应用层进行扩展，添加了四个扩展模块。

# 扩展功能设计

## pcap文件切分功能设计

pcap文件的报文使用字节码存储，所占用的存储空间很小。但对于解析后的报文数据而言，每条报文都包含大量对象、字段和方法，使得解析后的数据占用的存储空间是原始报文大小的数十倍。

由于解析pcap文件对内存造成极大的负担，而且内存占用率过高会影响程序的运行效率，那么，对于较大的pcap文件，将其切分成若干个小文件，再进行批处理非常必要。

本项目扩展了Scapy对大pcap文件进行切分的功能。开发人员只需提供所要切分的pcap文件路径、切分后的存储目录以及切分批大小即可按照默认设定简便地对pcap文件进行切分。同时开发人员还可自行设定以下切分过程中的需求：

1. 开发人员可以通过更改切分模式设置来设定切分模式。本项目的pcap文件切分功能提供两种切分模式供开发人员选择。第一种为兼容模式，调取文件解析模块解析数据后再进行切分，切分后可存储为与源文件不同的格式，但效率较低。另一种为极速模式，无需数据解析，对源数据进行切分，速度较快，但不可转换为与源文件不同的格式。默认为兼容模式。
2. 开发人员可以通过更改命名前缀设定来设置切分后的文件命名。默认情况下，新生成的文件命名方式为：源文件名\_00000~源文件名\_99999。当用户设定了命名前缀的情况下，新生成的文件命名方式为：前缀\_00000~前缀\_99999。
3. 开发人员可以设定报文数据的链路类型。本设置只有在切分模式为兼容模式时有效。
4. 开发人员可以设定切分后的报文的存储方式。开发人员可以选择报文字节码的存储方式是大端存储还是小端存储。本设置只有在兼容模式下有效。

兼容模式与极速模式时序图如图 4.1与图 4.2所示。

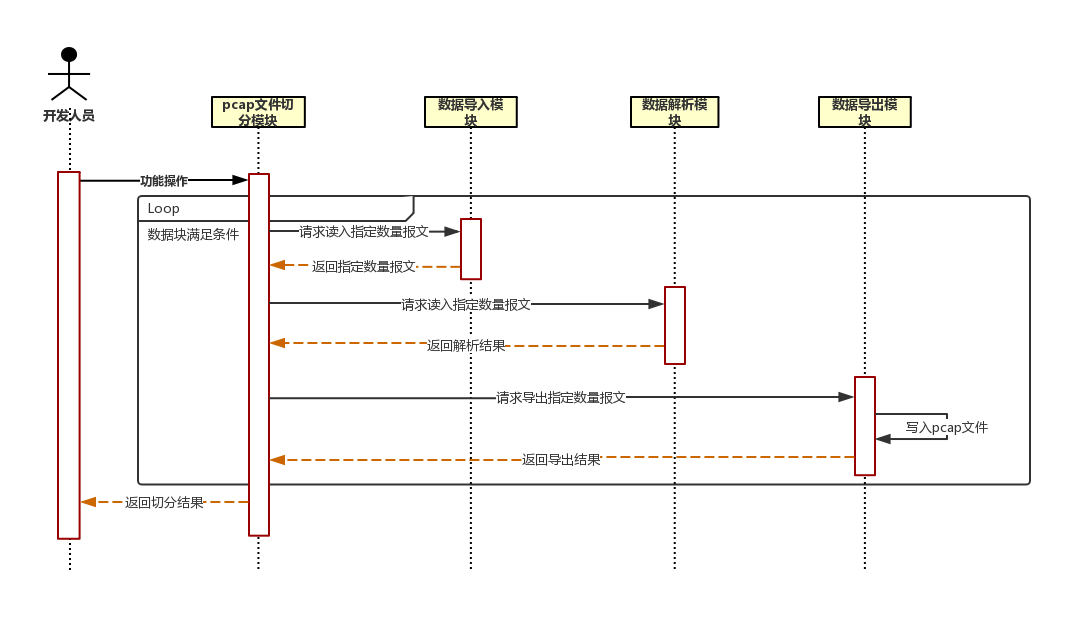


图 4.1 pcap文件切分兼容模式时序图

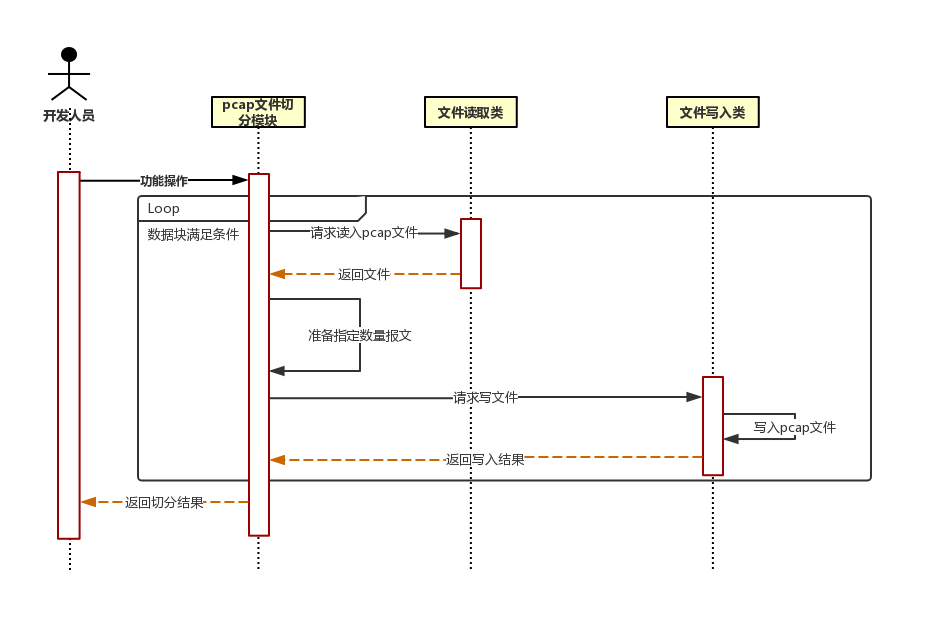


图 4.2 pcap文件切分极速模式时序图

## 获取报文协议类型功能设计

Scapy根据报文的协议层次将报文数据分层解析，解析的结果也按照协议层堆叠起来。解析结果的每一层是一个单独的对象，拥有一个payload变量，用于保存下一层协议的对象实例。

比如一条DNS报文pkt，其解析结果是一个物理层的Ether对象实例，pkt.payload是一个网络层的IP对象实例，pkt.payload.payload是一个传输层的UDP对象实例，依次类推。

在处理多协议流量数据时，用户通常需要对不同协议类型的报文进行不同的处理操作。但用户无法直接从Scapy解析的结果中获取报文的协议类型，必须自行设计一种判定逻辑来得到报文的协议类型。因此，为解析后的报文对象内置一个获取协议类型的函数可以为用户提供极大的方便。

本项目扩展了Scapy获取报文协议类型的功能，开发人员可以直接通过本功能来获得报文的协议类型。用户可以选择两种报文协议类型的返回方式：一种是返回单层协议类型，另一种是返回所有层协议类型。

同时当开发人员选择的返回类型为单层时，可以指定返回哪一层的协议类型。默认情况下返回可识别的最高层的协议。

时序图如图 4.3所示。

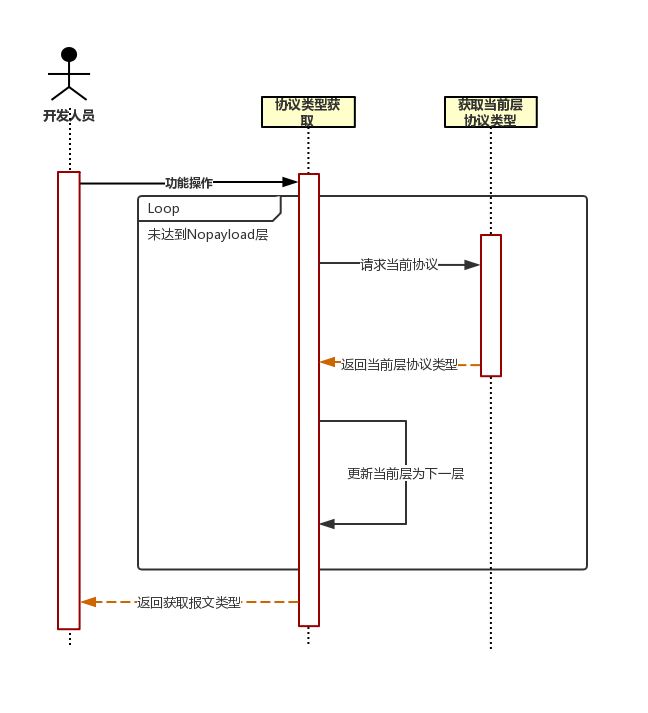


图 4.3 获取协议类型时序图

## 监听持久化

Scapy的监听存储策略为将监听到的所有报文都存在内存中，监听结束后再一起存储到硬盘上。由于pcap文件通常都很大，因此Scapy原有的监听存储功能很容易造成内存泄漏。另外，如果突然宕机，数据就会全部丢失。基于该情况我们利用内存快照设计了监听持久化模块。

监听持久化模块将截获到的报文放在缓存池中，并利用解析进程对缓存池中的报文进行解析，解析好的数据放入另一个缓存池中。用户可以根据需要将解析好的数据放入一个文件，也可以将数据按照时间或大小存成不同的文件。为了更好的满足用户的需求，我们设计了一下自定义的功能。在监听阶段用户可以自定义筛选规则、监听结束时间、监听报文数量以及停止监听的判定函数，详细说明如下：

1. 数据批大小：捕获数据时达到该阈值大小就对内存中数据进行一次存储写入
2. 监听端口：监听端口或端口列表，默认监听所有端口
3. 筛选规则：采用BPF规则，只监听符合要求的报文
4. 文件名前缀：结果文件名的前缀
5. 模式：是否采用极速模式，极速模式下不对监听到的数据进行解析
6. 停止监听的报文个数：当监听到用户指定的报文个数时，停止监听
7. 停止监听结束时间：到达用户指定的监听结束时间时，停止监听
8. 停止监听判定函数：用户可以自定义一个判定函数，当某条报文符合该判定函数的条件时，模块停止监听

监听持久化极速模式时序图如图 4.4所示，普通模式如图 4.5所示。

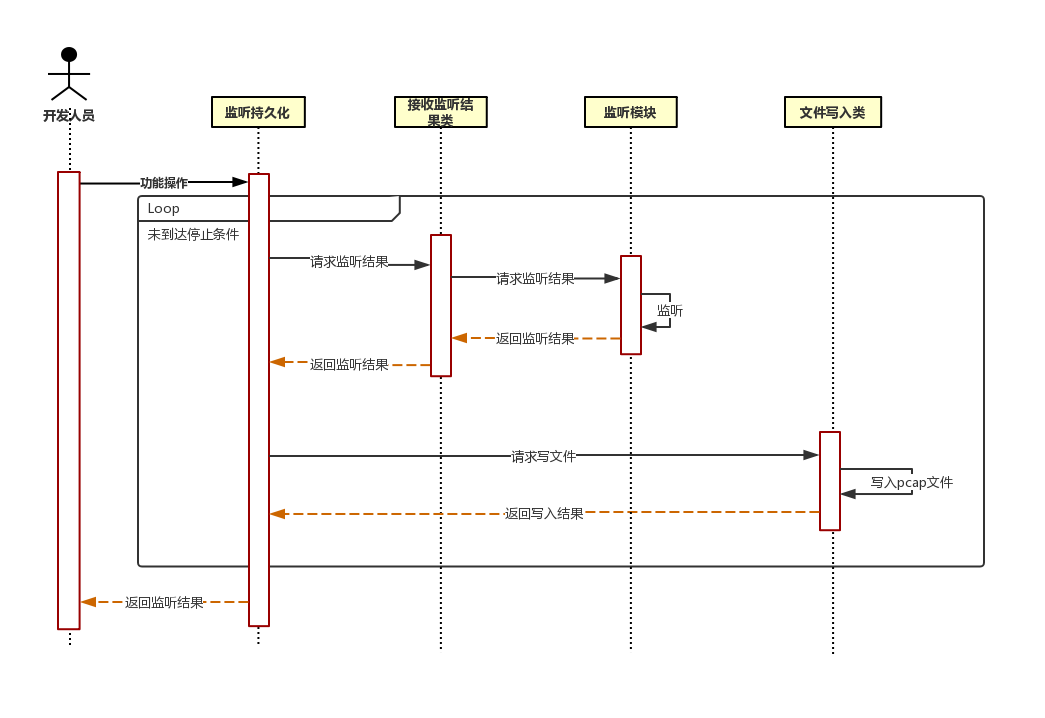


图 4.4 监听持久化极速模式时序图

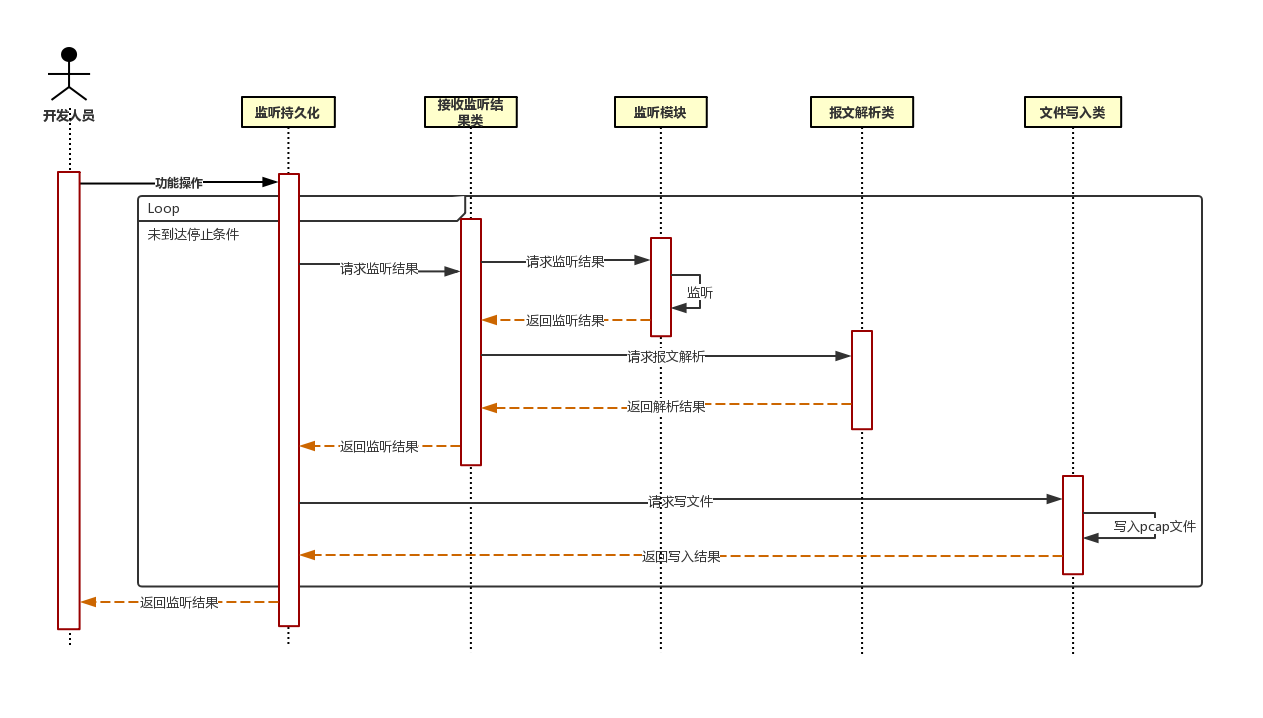


图 4.5 监听持久化普通模式时序图

## 会话提取功能

首先我们需要对会话的概念加以解释，此处的会话与HTTP中的会话有所区别。一次会话由互相通信的两个实体之间的交互过程组成。根据会话的通信性质，可将会话的两个实体分为客户端和服务端。一次会话过程由客户端和服务端的多次交互组成，并且客户端的一次请求总是对应于服务端的一次回应。图 4.1展示了客服端和服务器端的两次会话，在客户端向服务端发起TCP连接请求并被接受后，服务端和客户端开始了基于TCP和SMTP会话，图片中反映了客户端和服务端的多次交互过程。可以看到，由一个或多个相互交互的报文构成了会话，两次通信的时间间隔如果超过某一个阈值，将会认为是两次会话。

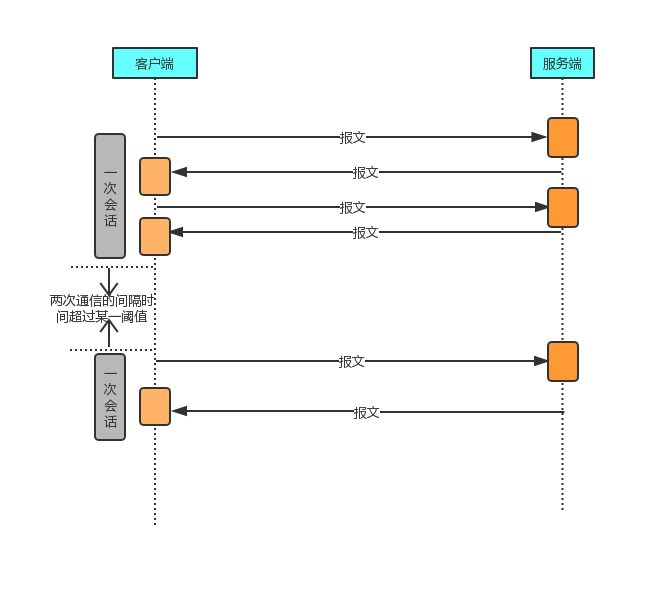


图 4.6会话示意图

一个pcap文件中包含大量会话，有时我们只需要其中一部分或是只需要从某一IP发往另一个IP的会话。基于这种应用场景，我们设计了会话提取功能：从指定的pcap文件中提取出符合特定条件的会话，并进行存储。提取的条件有两种：四元组提取和时间窗口提取，下面将分别介绍。

四元组提取指的是源IP地址、源端口、目的IP地址和目的端口四项条件，是最常用的会话的筛选条件。四元组的四个条件均为可选参数，即当参数不为空时根据传入的参数筛选符合条件的会话，当传入的参数为空时可以任意取值。

时间窗口提取是指根据用户设置的时间窗口大小将间隔时间在时间窗内的会话数据提取出来放入一个文件中。当用户做像Web服务流量监控的任务时，希望分析所观察到的传入数据流，采用时间窗口进行处理比较适当。应用时间窗之后的会话提取如图 4.2所示，一次会话不仅受到两次通信间隔时间的影响，用户设置的时间窗大小还会限制一次会话的最长时间。在实际应用中，用户可以根据应用场景调整窗口大小。

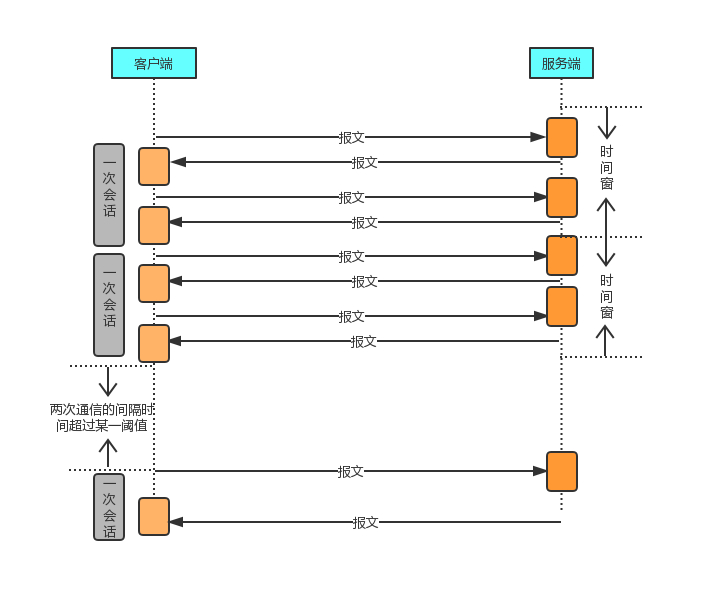


图 4.7 会话提取示意图

# 实现方案

## 文件切分功能

文件切分功能的主要作用是将大的pcap文件切分成一批小pcap文件，为了满足用户的不同的需求，本项目实现了两种切分模式：兼容模式和极速模式。

### 兼容模式

兼容模式下会对所有读取的报文进行解析，并且可以指定新pcap文件的部分参数，包括报文链路类型和数据存储方式。该模式的实现主要是借助Scapy内置的两个基层类PcapReader和PcapWriter来实现。

其中PcapReader类的主要功能是读取并解析pcap文件。在实例化时，该类会自动去读取pcap文件的头部数据，获取pcap文件的存储方式等全局信息。然后借助实例化后的PcapReader对象，就可以读取和解析数据块的字节码数据，从中获取到报文的全局信息和原始字节码数据，并对报文进行解析，返回解析后的报文对象。

PcapWriter类的主要功能是生成pcap文件，并写入报文数据。在实例化时，该类会记录下用户指定的全局参数，生成一个空的pcap文件，并保留文件句柄。在用户指定写入操作时，实例化后的PcapWriter对象会先使用文件句柄写入文件的全局信息，然后再将解析后的报文封装成数据块写入其中。

兼容模式的流程图如图5.1所示。

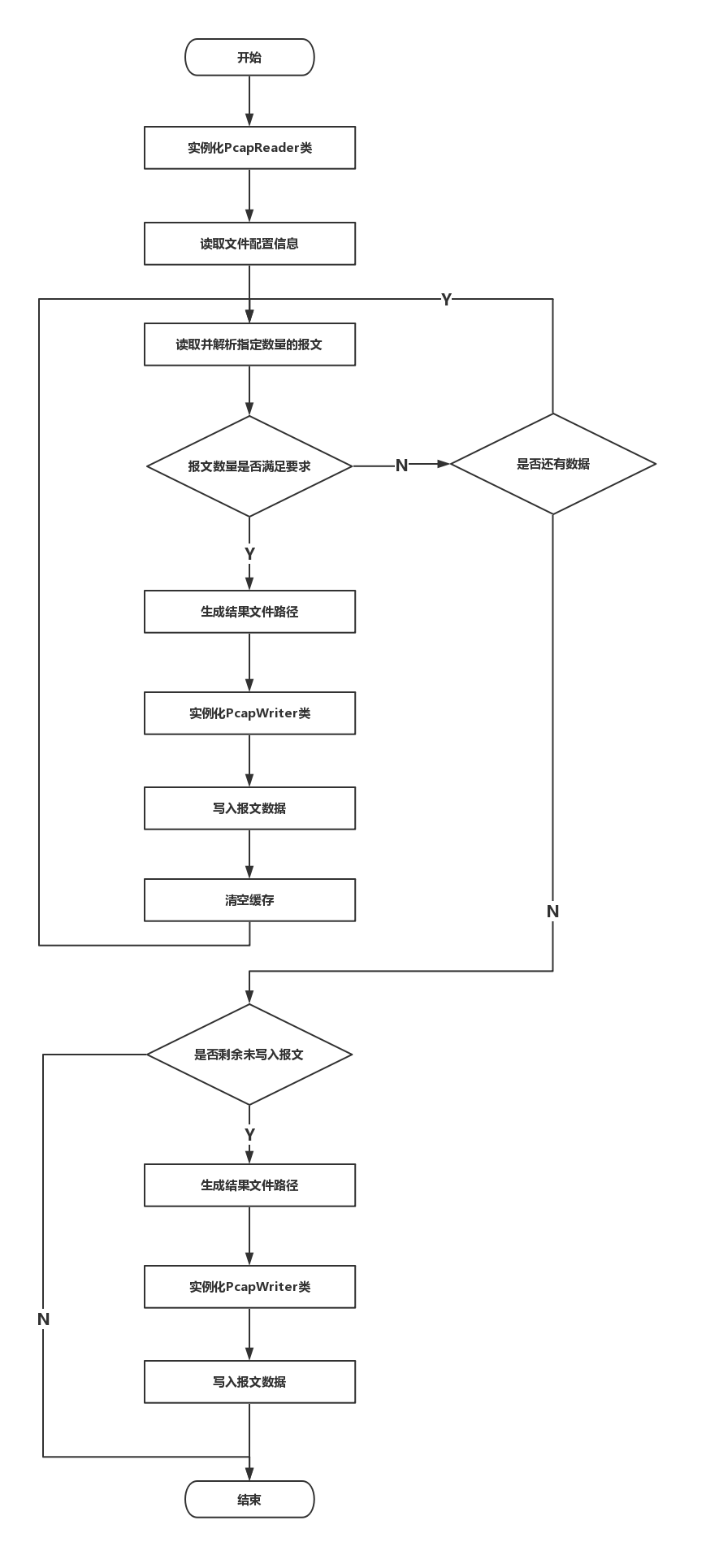


图 5.1兼容模式流程图

### 极速模式

极速模式采取的是一种“拷贝”策略，对于读取到的报文数据不进行解析，也无法指定新pcap文件的全局参数。这种方式虽然会降低文件切分的灵活性，但会极大的提高文件切分效率。

pcap文件由文件头和若干数据块组成，文件头共有24个字节，指明了本文件的存储方式、最大存储长度等内容；每个数据块都包含块头和块体两部分，块头共16字节，指明了报文的截获时间与报文长度，块体则是原始的报文字节码数据。pcap文件的格式如图5.2所示。

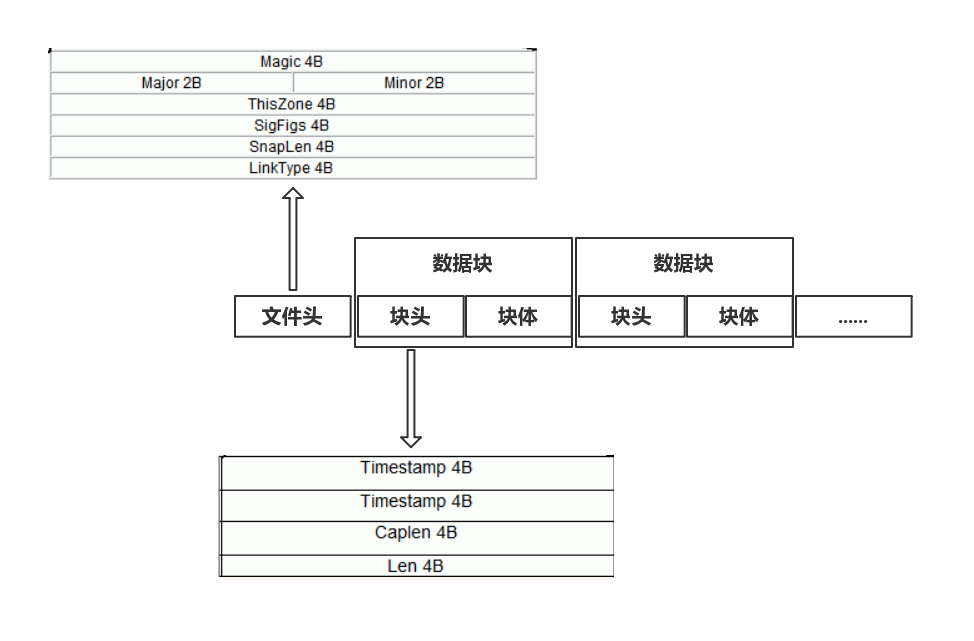


图 5.2 pcap文件格式

为了实现极速模式，本项目专门封装了两个新的文件读取和写入类。文件读取类在实例化时会自动读取前24个字节的文件头字节码，并保存在内置参数中，然后只解析文件存储方式字段Magic。同时，该类提供了两个内置功能，一是获取文件头的原始字节码，二就是获取一个数据块的字节码。在读取数据块时，先读取16字节的块头数据，解析其中的Caplen字段，然后读取Caplen字段指定字节数的块体数据。文件写入类在实例化时也会生成一个空的pcap文件，并保留文件句柄。然后由用户提供文件头和数据块的原始字节码数据，该类只负责将这些数据直接写入文件中。

极速模式的流程图如图5.3所示。

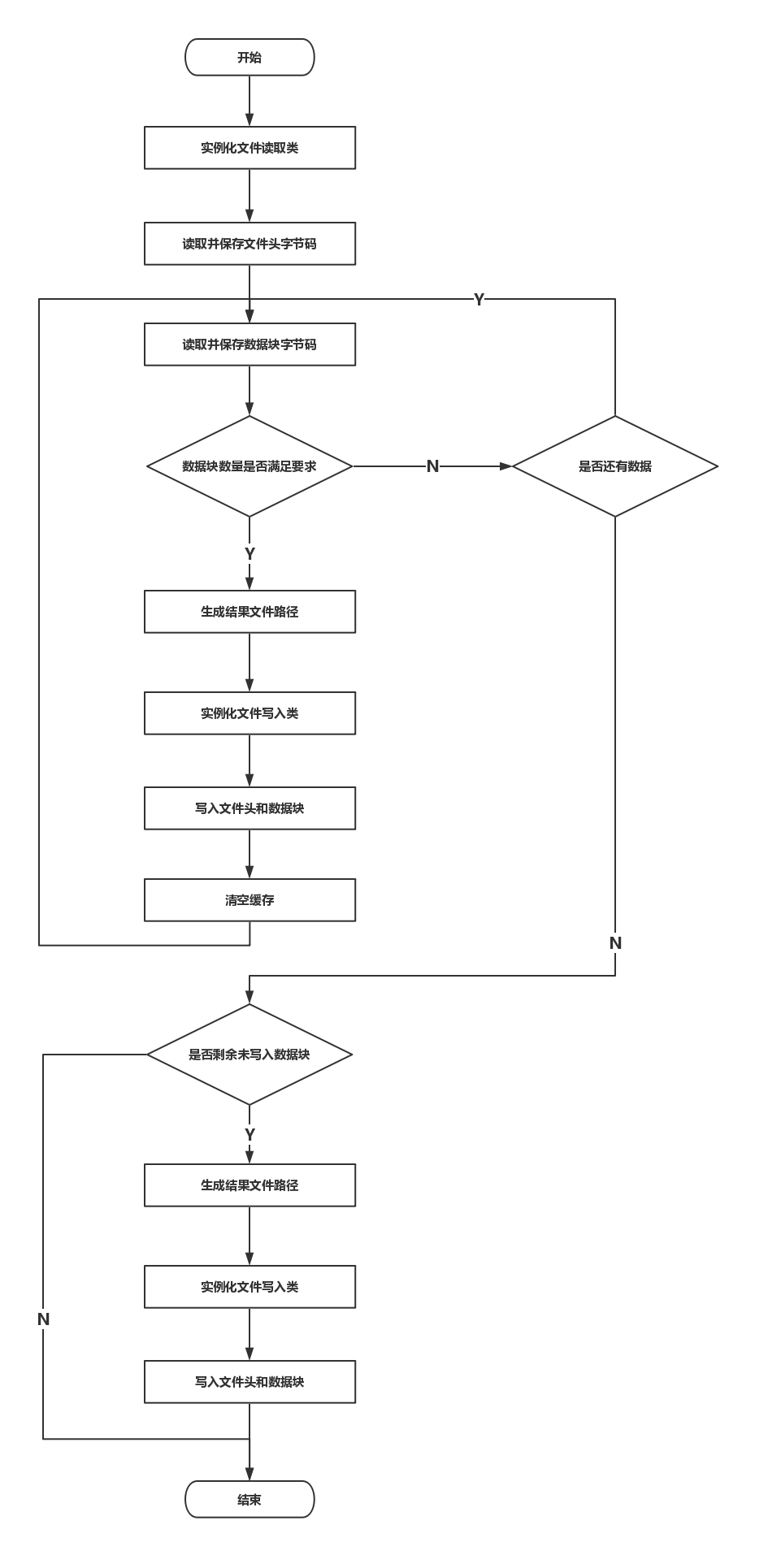


图 5.3极速模式流程图

## 监听持久化功能

在进行大规模流量监听时，有两个需要解决的问题：一是监听到的数据如何存储，二是如何提高处理速度。对于第一个问题，Scapy本身是将监听到的数据全部保存至内存中，在面对大流量时很容易发生内存泄漏。因此本扩展部分第一个实现的功能就是将监听到的数据分批存储到文件中，及时释放内存。对于第二个问题，Scapy本身会对监听到的报文进行解析，而Scapy的解析是非常耗时的。在监听大规模流量时，会导致处理速度赶不上监听速度，缓冲区发生溢出，部分报文数据就会缺失。因此本扩展部分第二个实现的功能就是在监听时不解析报文，并封装了新的数据结构与pcap文件写入功能。

本部分的主要工作有以下三点：

1. 参考原有的监听函数重新封装一个监听持久化函数，去掉无用的参数与功能。
2. 添加新的参数与存储功能。
3. 设计新的数据结构，修改底层端口监听类SuperSocket，增加只监听不解析的功能，并封装与新数据结构相应的pcap文件写入类。

第一部分工作主要是将与本功能无关的参数都删去，相关的功能代码也都一一删除。第二部分的工作主要是借助pcap文件写入类，在监听的同时，将监听到的报文数据持续的写入pcap文件中，及时释放内存。

第三部分工作比较麻烦，本功能不再解析报文，因此需要重新设计一种数据结构，至少要包含pcap文件写入时的必有信息，包括但不限于报文原始字节码、报文截获时间等。其次，Scapy的监听功能使用的是底层一系列的Socket类实现的，通过recv函数获取并解析报文。因此，本部分还需要在最顶层的父类SuperSocket中封装一个recv\_noanalysis函数，该函数只获取监听到的报文，然后根据设计好的数据格式封装报文，取消了解析操作。最后是报文数据的存储，因为报文数据的格式不同了，因此需要重新封装一个pcap文件写入类，该类要支持文件头部写入、新格式报文数据的写入以及持续写入数据功能。

监听持久化功能的流程图如图 5.4所示。

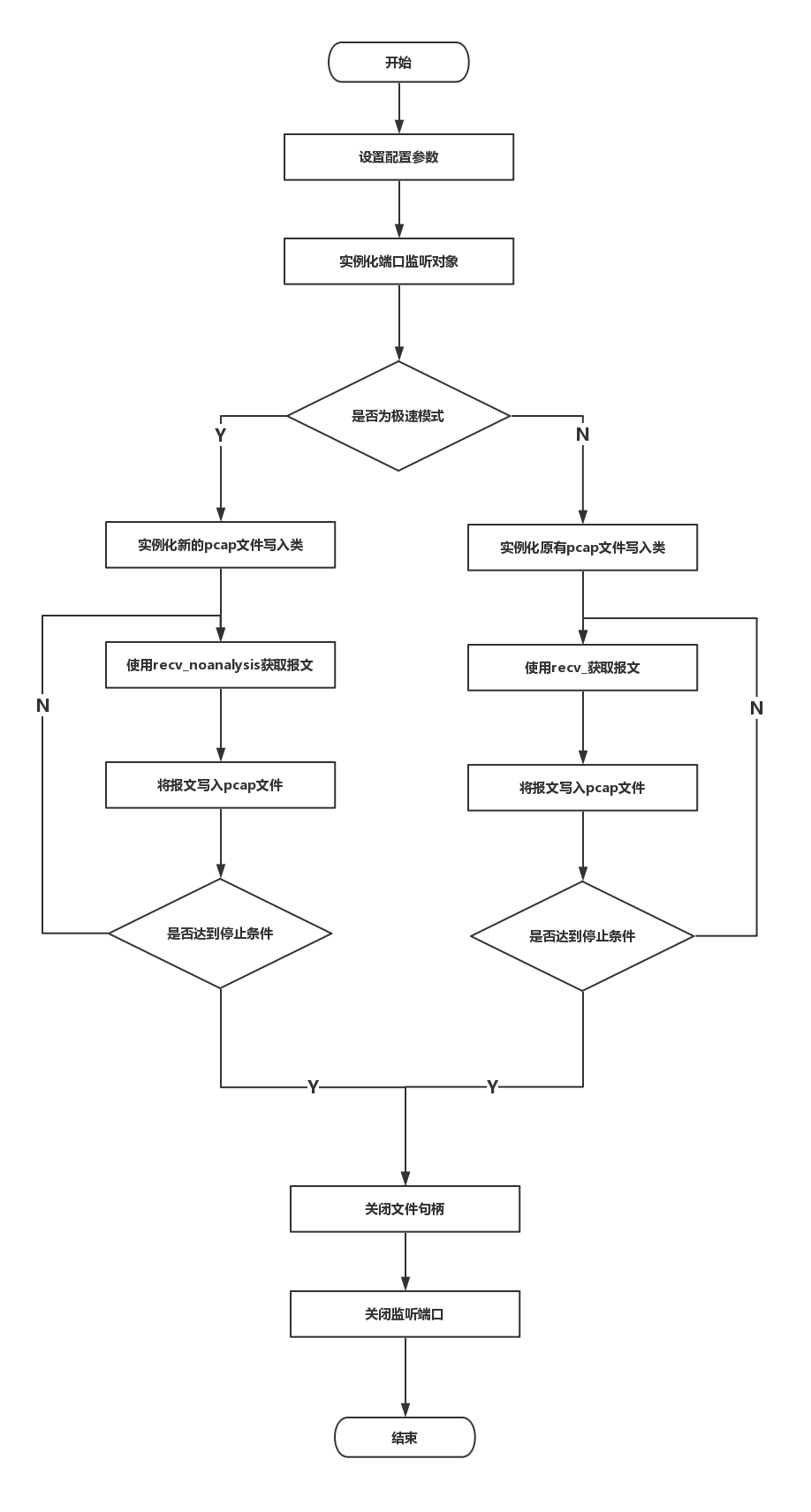


图 5.4监听持久化流程图

## 协议类型获取功能

协议类型获取功能主要作用是让用户更方便获取报文各层协议类型，在实现的过程中，主要操作除了获取每层的对象名之外，还需要判断哪一层可以代表整个报文。

在Scapy中，报文对象的各层可以是Scapy能够解析的各种协议类型，比如Ether层、IP层、TCP层等，也可能是某种协议携带的填充数据Padding层或者是Scapy暂时不支持的协议Raw层或者是一个空层NoPayload层。因此，在选择可以代表整个报文的协议名时，不能简单的选择最高层，而是要考虑到上述的所有情况。

协议类型获取功能的流程图如图 5.5所示。

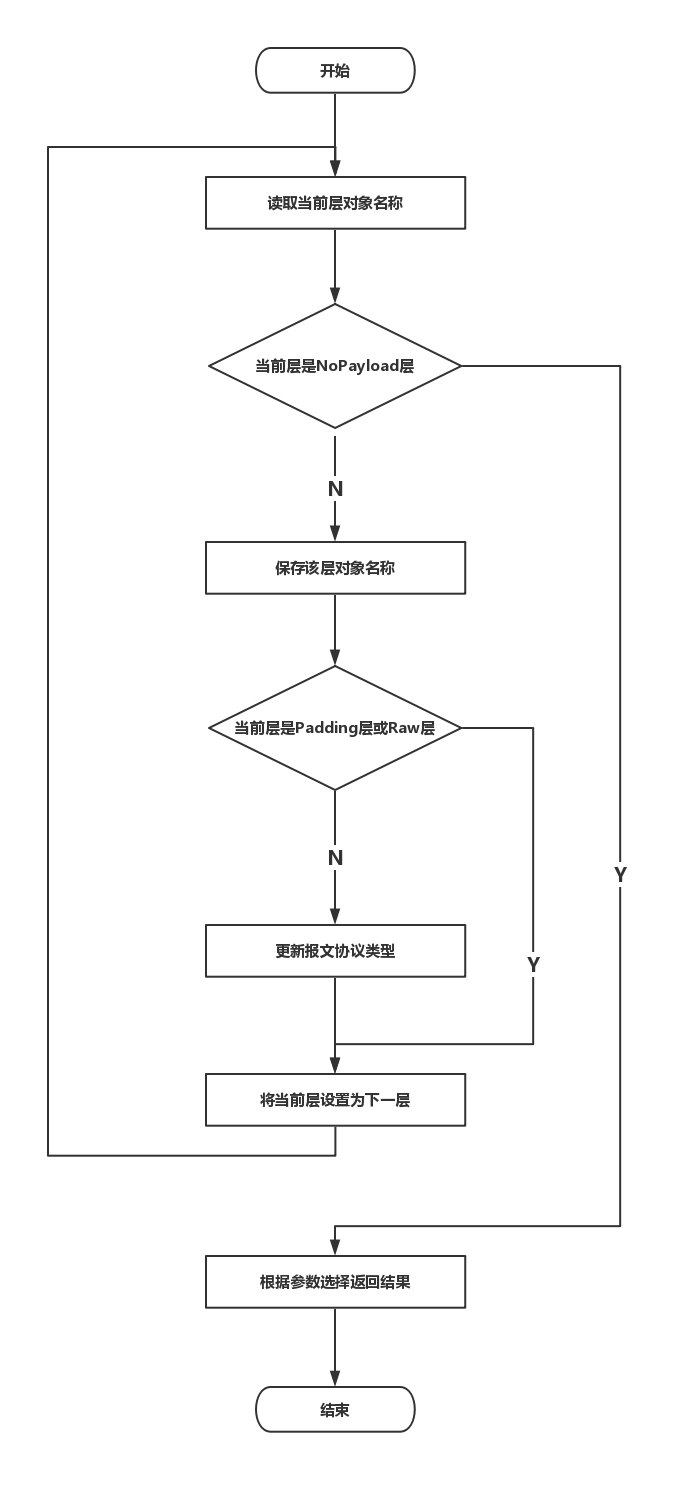


图 5.5协议类型获取功能流程图

## 报文会话提取模块

### 基本报文会话提取流程实现

报文会话提取功能主要是将从pcap数据中的数据包按照四元组（源IP，目的IP，源端口，目的端口）和时间窗口划分为多个独立的“会话”，并以文件形式进行存储。

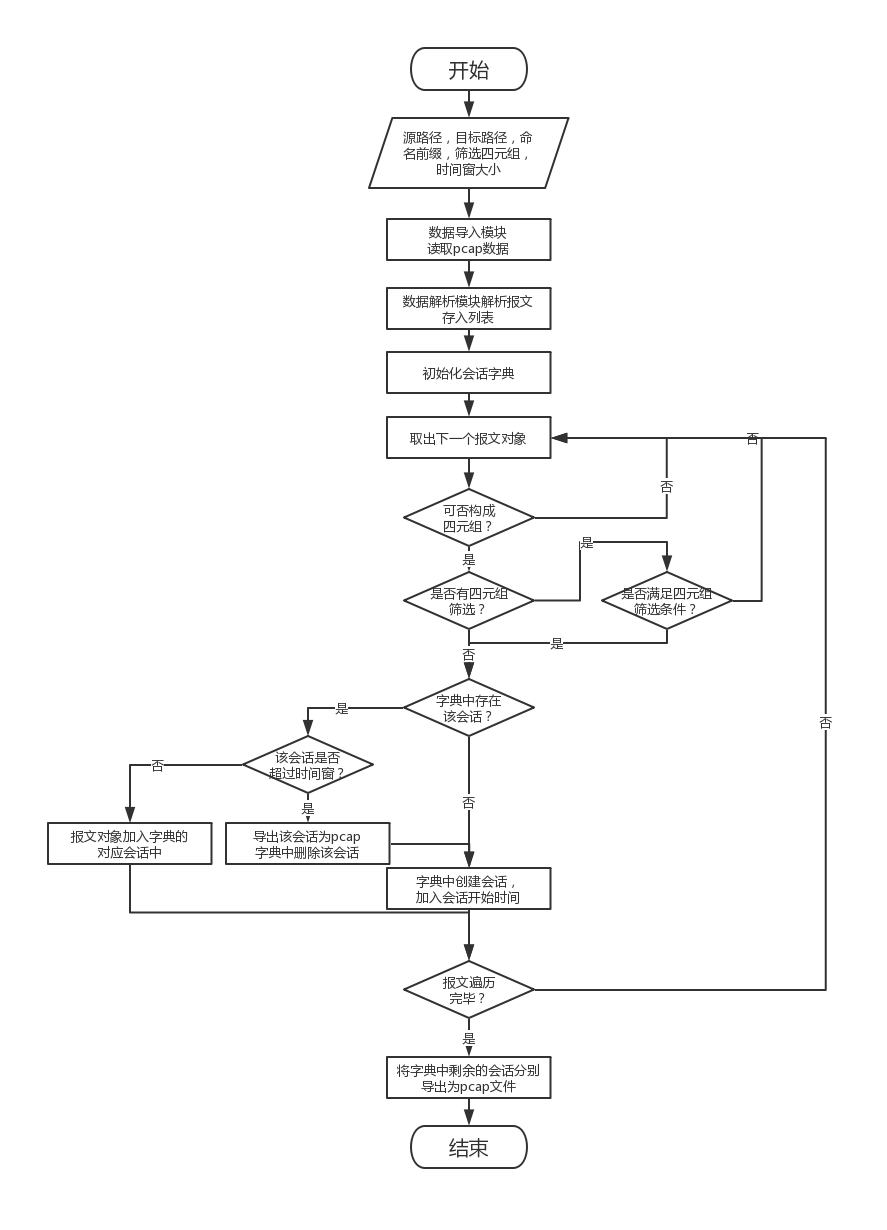


图 5.6 报文会话提取模块主要流程

报文会话提取模块的主要流程如图 5.6所示，首先需要模块调用者提供需要提取的源数据路径，目标文件存储目录，可选择提供命名前缀，筛选四元组和时间窗大小。接下来模块调用Scapy的数据导入模块导入pcap数据，然后调用数据解析模块对导入的数据进行解析，解析后的分组报文将被加载到一个列表中。

为了能够对分组报文归入对应的“会话”，需要初始化一个空的会话字典。接下来对列表进行遍历，取出下一个分组报文，判断报文网络层和传输层可否构成四元组，若否则跳过，继续处理下一分组。若是，则根据提供的四元组筛选条件进行验证，未通过则跳过处理下一分组。在会话字典中查找到当前分组对应的会话项（键值对），若不存在，则新建该会话项，并加入当前的分组报文和会话创建时间。否则对已有的会话项到当前分组报文的时间距离和时间窗进行比较，若在时间窗内，加入当前分组报文，若不在时间窗内，先将字典中该会话项存储为pcap文件，删除后再新建会话项。然后继续处理下一分组直至列表遍历完成。最后，将会话字典中剩余的会话项各自存储为对应的pcap文件。

另外，由于提取的会话是某个IP的一个端口和另一个IP的一个端口进行一段通信的数据，会存在请求和回复报文，因此四元组的筛选匹配是不区分方向的（从源地址到目的地址的方向性），并且对于四元组中的IP和端口可进行任意性匹配，例如（192.168.99.1，x.x.x.x，22，yy）可以提取出和192.168.99.1:22进行通信的分组报文会话，（x.x.x.x，y.y.y.y，80，zz）可以提取出所有和80端口相关的会话。

### 分批会话提取流程实现

对于较大的pcap源数据，一次性读入然后进行会话提取对于内存空间的占用量很高，为了控制对内存空间的占用量，这里采取分批读入的方式读取pcap数据进行解析，然后对这批分组报文进行之前同样的会话提取处理。在每批数据处理完之后，为了及时对到达时间窗的会话进行存储并释放空间，用最后一个分组报文的时间对会话字典中所有的会话项进行一遍刷新，释放超过时间窗的会话项，而未超过时间窗的会话项得以保留，在所有的批次处理完之后，再对字典中剩余的会话项进行统一结算存储。

考虑到一般会话的长度，默认时间窗为300s，这是为了防止时间窗过小而截断正常的一段会话，但实际报文中存在大量非常短小的会话，在基础方案实现下，短小会话的四元组若不再出现，则该会话项很可能将会在内存中保存多个批次。因此，此处提供另一种时间窗划分方式，即时间窗规定为会话中两个报文分组最长间隔时间，这样可及时清算掉不活动的会话。

分批会话提取流程主要步骤如图 5.7所示。

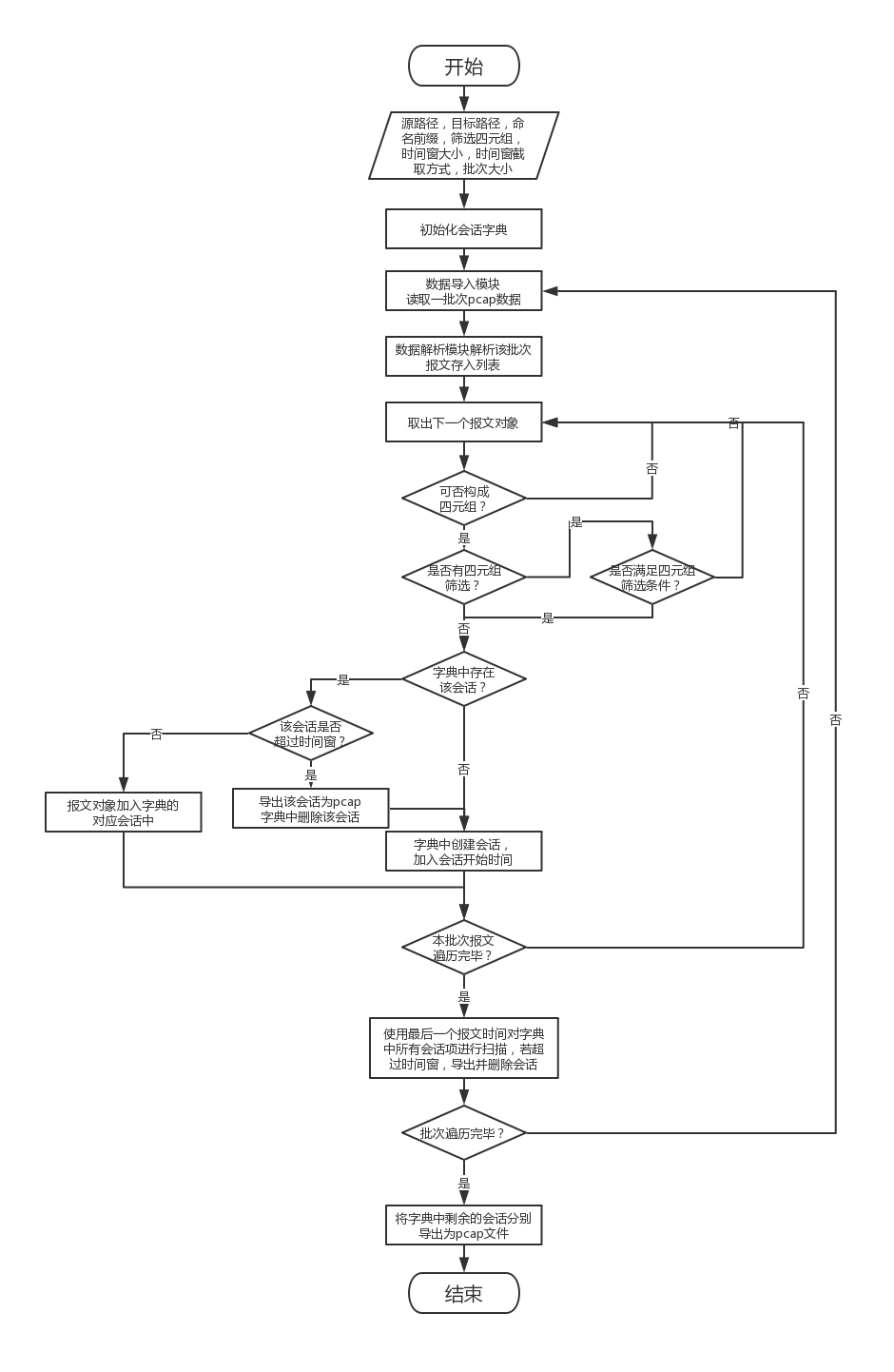


图 5.7 分批会话提取流程

# 集成方案

## 集成方式

Scapy是一个Python工具库，集成扩展功能非常方便，只需要将扩展部分代码放入相应源文件中，即可在代码中或者控制台交互界面中调用。

比如pcap文件切分功能的相关代码可以直接放到utils.py文件中，该文件封装了Scapy本身自带的所有数据导入和导出相关功能代码。监听持久化功能的相关代码可以直接放到sendrecv.py文件中，该文件封装了Scapy本身自带的所有数据监听和发送的相关功能代码。当然，有些情况需要修改Scapy本身的代码，也是可以在相应文件中直接修改的。

当需要在其他机器上使用扩展后的Scapy时，只需先按官方推荐的方式安装好相应版本的Scapy，然后将本项目扩展后的代码覆盖掉整个原来的Scapy代码即可。

## 验证样例

本节将使用一个测试函数scapy\_extend\_test( test\_object )来验证扩展部分代码的集成方案是可行的。

测试函数如图 6.1所示：

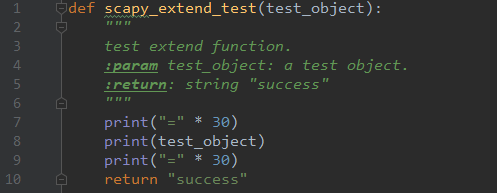


图 6.1 集成测试函数

将该函数添加到文件scapy/utils.py中，交互式命令行中运行结果如下：

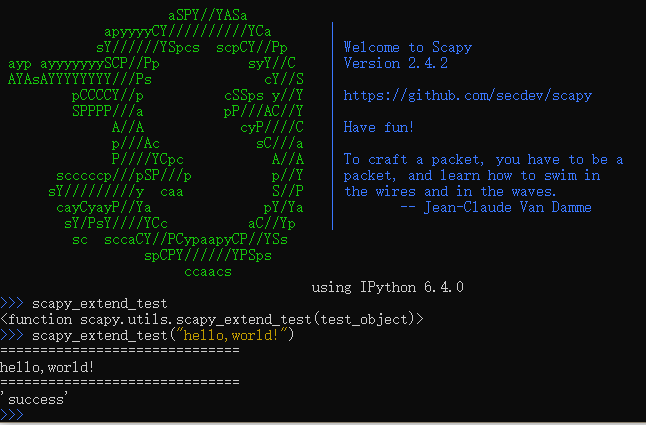


图 6.2 交互式模式测试运行结果

作为Python库导入测试如下：

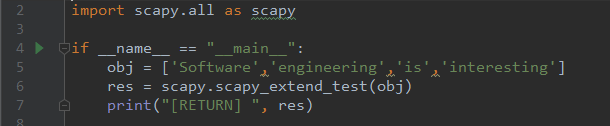


图 6.3 测试代码

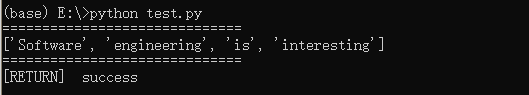


图 6.4 Python库导入下测试结果

在scapy项目下新建一个mytest.py，并将测试函数加入，然后在scapy/all.py中加入如下代码：from scapy.mytest import \*

在交互式模式和Python库导入下测试结果如下：

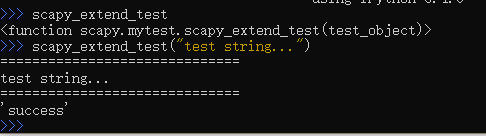


图 6.5 命令行下mytest测试结果

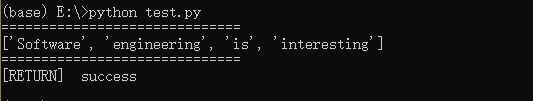


图 6.6 Python库导入下mytest测试结果

结果表明，Scapy下该扩展部分代码集成方案是可行的。